Betriebssysteme Zusammenfassung

Maximilian Wolf

July 31, 2025

Contents

1 A Organisation

1.1 Aufgaben eines Betriebssystems

- Verwalten von Anwendungen
- (Grafische Oberfläche)
- Ressourcenverwaltung
- Statische und dynamische Strukturierung von Programmsystemen
  - Prozessor
  - Hauptspeicher
  - Hintergrundspeicher
  - Dateien
  - Fairness
  - Effizienz

2 C Aufbau eines Rechnersystems

2.1 Hauptspeicher

- Adresse mit fester Bitbreite zur Auswahl einer Zelle
- Zellengröße ein Byte, Zugriff jedoch häufig Wortweise

2.2 Prozessor

- Register zur Speicherung von Zwischenergebnissen / Adressen
- Program Counter (PC) speichert Adresse für nächste Instruktion

2.3 Befehlsbearbeitung

- Laden der nächsten Instruktion
- Interpretation des Befehls
- Ausführen des Befehls
- Inkrementieren des PC

2.4 Start des Systems

- Initialisierung PC
- Initialisierung ROM mit Programm
- Initialisierung der Hardware
- Laden und Starten des Beriebssystems

2.5 Abstraktionsebenen

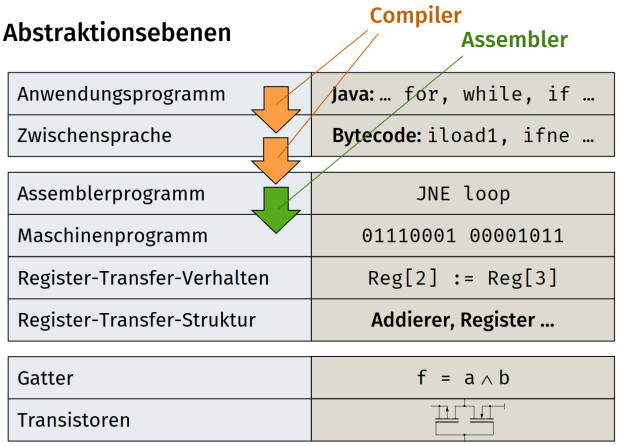

Figure 1: Abstraktionsebenen am Beispiel von Java

3 D Einführung in Betriebssysteme

3.1 Betriebsarten

- **Stapelverarbeitung** (Batch Processing) - Bearbeitung nacheinander
- **Interaktiver Betrieb** (Interactive Processing) - Sofortige Reaktion auf Befehle
- Time-Sharing Betrieb - Aufteilung über mehrere Nutzer / Programme
- **Echtzeitbetrieb** (Real-Time Processing) - Reaktion innerhalb fester Zeitschranken
- **Ein-/ Mehrprozessbetrieb** (Singletasking / Multitasking)
- **Ein-/ Mehrbenutzerbetrieb** (Single-User, Multi-User)

3.2 Verwaltung von Ressourcen

3.2.1 Virtuelle Ressourcen

- Speichersegmente
- Dateien
- Semaphore

3.2.2 Aufgaben

- Belegung von Ressourcen auf Anforderung
- Aufteilung Ressourcen für Benutzer / Anwendungen
- *Schutzumgebung* (Wechselseitiger Ausschluss des Ressourcenzugriffs, Zugriffsberechtigungen)

3.2.3 Klassifizierung von Ressourcen

- aktiv, zeitlich aufteilbar (z.B. Prozessor)
- passiv, exklusiv nutzbar (z.B. Drucker)
- passiv, räumlich nutzbar (z.B. Hauptspeicher, Hintergrundspeicher)
- Unterstützung bei Fehlerbehandlung

3.3 Betriebssystemkern (Kernel)

Aufgaben des Betriebssystemkerns:

- Anwendungs-/ Prozessverwaltung
- Dateisysteme
- Speicherverwaltung
- Verwaltung von Ein-/ Ausgabegeräten
- Gerätetreiber
- Systemaufrufe (System Calls)

3.4 Betriebsmodi eines Prozessors

Repräsentation durch CCR Flag 0 = User-Mode, 1 = Supervisor-Mode

- Benutzermodus (User-Mode)
  - Anwendungen
  - eingeschränkter Befehlssatz
- Privilegierter Modus (Supervisor-Mode)
  - Betriebssystem
  - uneingeschränkter Befehlssatz
    →erlaubt Konfigurationsänderung des Prozessors, Wechsel des Modus, spezielle I/O Befehle

3.5 Unterbrechungen

3.5.1 External Interrupts

Signalisierung durch IRQ (Interrupt Request) **Interrupt Service Routine**

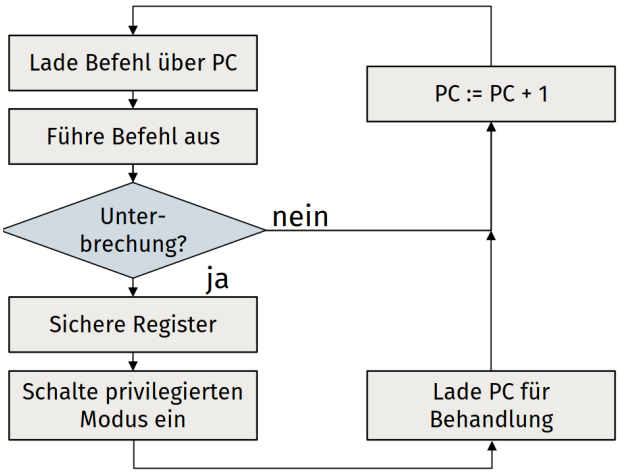

Figure 2: External Interrupt

(Reaktion auf externes Ereignis)

- Prozessor unterbricht laufende Bearbeitung, Sicherung der Registerinhalte
- Ausführung der definierten Befehlsfolge im *privilegierten Modus*

- Wiederherstellung des ursprünglichen Zustands durch Laden der gesicherten Registerinhalte

Beispiele:

- Fehlerbedingung
- Ankommende Netzwerknachricht
- Rückmeldung durch langsame Geräte
- "Wecker" Funktion

3.5.2 Internal Interrupts / Exceptions

Fehlersituationen z.B. Division durch Null, Ausführungsversuch eines privilegierten Befehls im User-mode

- Unterbrechung des Befehlsfluss, Sprung in Unterbrechungsbehandlung
- Rückkehr zur bisherigen Befehlskette mit Return from Interrupt (RTI)

3.5.3 User Interrupt / Trap

Übergang von User-Mode zu Supervisor-Mode
Systemaufruf, System Call, Supervisor Call

- Spezielle Befehle zum Eintritt in Privilegierten Modus
- Aktivierung des *Privilegierten Modus*
- Ausführung der Befehlsfolge
- Schnittstelle zum Betriebssystem

4 E Prozessverwaltung und Nebenläufigkeit

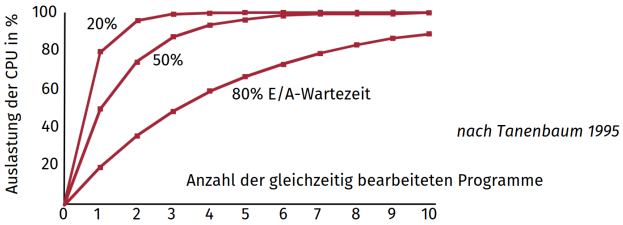

Figure 3: Wartezeit nach CPU Auslastung

4.1 Terminologie

- **Programm**: Folge von Anweisungen und dazugehörige Daten
- **Prozess**: Programm in Ausführung inklusive Daten, Mehrfachausführung möglich
- **Aktivitätsträger**: Befehlsstrang eines Prozesses
- **Prozessor**: Rechenwerk, welches Anweisungen eines Aktivitätsträger ausführt

4.2 Prozesse

4.2.1 Bestandteile

- Speicher (Instruktionen und Daten)
- Ein / Mehrere Aktivitätsträger (Befehlsbearbeitung) →Virtueller Prozessor
- Kontext
  - Prozessor: Registerinhalte inkl. Programmzähler
  - Ressourcen: Verwaltungseinheit für Dateien, Semaphore, Queues, Pipes, Sockets, Geräteverbindungen
- Schutzumgebung
- Prozesskontrollblock / Process Control Block (PCB)

4.2.2 Prozesskontrollblock

- Prozessnummer / Process Identifier (PID)
- Metadaten (Rechenzeit, Erzeugungszeitpunkt)
- Eigentümer (User Identifier [UID], Group Identifier [GID])
- Kontext: Register inkl. PC
- Speicherabbildung (Speicherbereiche, Schutzumgebung)
- Arbeitsverzeichnis
- Verwendete Ressourcen

4.3 Erzeugung von Prozessen

Zuerst wird durch einen *Fork*-Aufruf der aufrufende Prozess kopiert und durch einen *exec*-Aufruf Umgebungsvariablen und Argumente für einen neuen Prozess übergeben.

4.3.1 Fork

Erzeugung eines Prozess (Child) durch einen anderen (Parent)
Prozesse haben dieselben Informationen

4.3.2 Exec

Prozess startet mit anderem Programm
Start durch Shell oder Benutzeroberfläche
Übergeben werden:

- Name-Wert-Paare, typischerweise zur Konfiguration
- Liste von Strings als Parameter (Java: void main (String args[]))

4.4 Kontextwechsel (Umschaltung von Prozessen)

- Sicherung des Kontext (Register), inkl. PC
- Laden des Kontext des anderen Prozesses (Register)

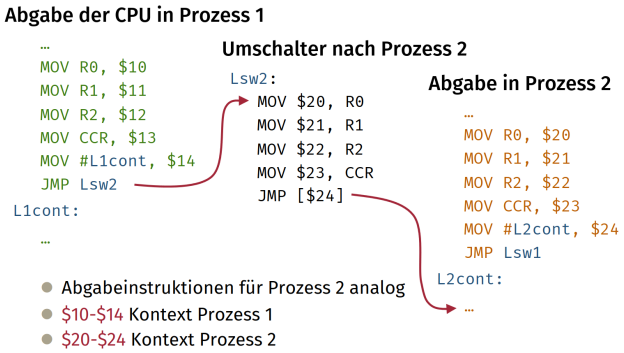

Figure 4: Kontextwechsel Beispiel

4.4.1 Kooperatives Multitasking

Prozesse geben freiwillig Prozessor ab
Beispiele:

- yield: Systemaufruf zur Abgabe des Prozessors
- Co-Routine: Abgabe des Prozessors an anderen

4.4.2 Scheduler

Verwaltung des aktuellen Prozesses, Umschaltung nach Strategie

- Systemaufruf: Freiwilliger Wechsel
- Unterbrechung: Unfreiwilliger Wechsel

4.5 Prozesszustände

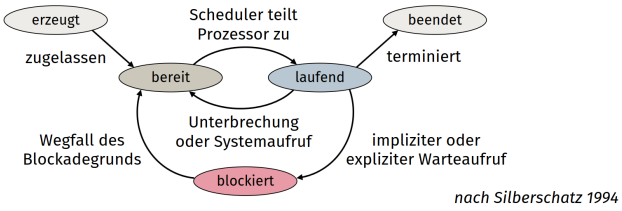

Figure 5: Prozesszustände

- **Erzeugt** (New) Prozess ist erzeugt, hat aber noch nicht alle Ressourcen
- **Bereit** (Ready) Prozess ist bereit zum Laufen
- **Laufend** (Running) Prozess wird ausgeführt
- **Blockiert / Wartend** (Blocked / Waiting) Prozess wartet auf Ereignis (z.B. I/O, Zuteilung Ressource, Empfang Nachricht)
- **Beendet / Terminiert** (Terminated) Prozess ist beendet, Ressourcen aber noch nicht freigegeben, andere Gründe z.B. Status muss Vaterprozess bekannt gegeben werden

4.6 Auswahlstrategien

- **Präemptiv**/Preemptive (Verdrängend): Unfreiwillige Abgabe des Prozessors
- **Nicht-präemptive**/Non-preemptive (Nicht Verdrängend): Freiwillige Abgabe des Prozessors

4.6.1 First Come, First Served (FCFS)

- Warteschlange für Prozesse im Zustand **bereit**
  Bei Blockierung eines Prozesses: Einreihung an das Ende der Warteschlange
- Bei lang laufenden Prozessen: "Konvoi-Effekt" (Stau), nicht für Time-Sharing oder interaktiven Betrieb geeignet

4.6.2 Shortest Job First (SJF)

- Auswahl des auszuführenden Prozesses nach Rechenzeit bis zur nächsten Warteoperation
- Vorhersage der Rechenzeit durch Protokollierung (Mittelwert, exponentielle Glättung)
  Im Vergleich zu First Come, First Served (FCFS) optimierte mittlere Wartezeit

Varianten Präemptiv, Nicht-Präemptiv (4.6)

4.6.3 Highest Priority First (HPF)

- Reihenfolge der Ausführung basiert auf Priorität, SJF ist z.B. HPF, da höhere Rechenzeit entspricht geringerer Priorität
- **Prioritäten** Statisch, Dynamisch z.B. SJF dynamische Prioritäten
- **Starvation** (Aushungerung): Niederpriore Prozesse werden von höherprioren verdrängt und nie ausgeführt
  →**Aging** (Alterung): dynamische Anhebung der Priorität für lang wartende Prozesse

Varianten Präemptiv, Nicht-präemptiv (4.6)

4.6.4 Round Robin (RR)

FCFS (4.6.1) mit automatischer Verdrängung nach Zeitintervall, nach Ablauf wird Prozess erneut in Warteschlange eingereiht

4.6.5 Multi-Level Feedback Scheduling (MLFQ)

- **Erste Ebene** präemptives HPF (4.6.3)
- **Zweite Ebene** Schedulingklassen mit RR (4.6.4) Strategie
- Wechsel der Schedulingklasse nach: *Verdrängung, Deblockierung, Alterung*

Beispiele siehe Folien E.71-E.77

4.7 Prioritätenumkehr

Gegeben seien Prozesse: L (niedrige Priorität), M (mittlere Priorität), H (hohe Priorität)
Problem

- L sperrt Ressource, die H benötigt →H muss warten
- M wird aufgrund der höheren Priorität als L ausgeführt, jedoch wird H dadurch länger blockiert, obwohl dieser höhere Priorität hat

Lösung: *Priority Inheritance*: L erbt vorübergehend Priorität von H, danach wieder niedrigere Priorität

4.8 Statische vs. dynamische Prioritätsvergabe

- **Statisch**: Priorität ist zur Laufzeit festgelegt
- **Dynamisch**: Priorität kann sich während der Ausführung ändern, z.B. durch Aging oder Deadlines

4.9 Aktivitätsträger (Threads)

Vorteile von mehreren Threads pro Prozess

- Aufteilung auf mehrere gleichzeitige Aufgaben, Ausnutzung von Mehrkernprozessornen- / systemen
- Prozess dient als Hülle für gemeinsame Ressourcen, gemeinsame Schutzumgebung und gemeinsam genutzte Dateien, Netzwerkverbindungen etc.

4.9.1 Kernel-level Threads

- Werden vom Scheduler des Betriebssystems verwaltet
- *Fairness* wichtig bei Scheduling-Strategie
- Basis für *User-Level* Threads
- Systemaufrufe zum Erzeugen, ändern der Prioritätsebene, Anhalten bzw. Löschen von Threads
- Arbeiten unabhängig voneinander und sind echt parallel

Beispiele: *POSIX-Threads* bei UNIX, *TPL Threads* bei Windows, *Java Threads*: User-Level Threads, die direkt auf Kernel-Level threads abgebildet werden

4.9.2 User-level Threads

- Werden auf Kernel-Level Threads abgebildet
  →Eingeschränkte Parallelität
- Können sich gegenseitig beeinflussen (blockieren)
- Umschaltung erfolgt schneller als bei Kernel-level Threads, da kein Wechsel ins BS notwendig
- Scheduling Strategie beliebig

Beispiele: *Java Virtual Threads, Windows Fibers*

4.10 Nebenläufigkeit und Parallelität

4.10.1 Nebenläufigkeit

- Nebenläufige Aktivitätsträger arbeiten unabhängig voneinander, kein Watzen bzw. keine Synchronisation erforderlich
- Umschaltung durch Scheduler, daher langsamer als parallele Threads

4.10.2 Paralellität

- Gleichzeitige Ausführung der Anweisungen mehrerer Aktivitätsträger
  →Auszuführende Anweisungen müssen unabhängig voneinander sein
- Nur auf Mehrprozessor / Mehrkern/- systemen möglich

4.10.3 Koordinierung

- Einschränkung der Nebenläufigkeit, um gleichzeitigen Zugriff auf gemeinsame Ressourcen zu verhindern
  →Einschränkung zeitlicher Durchmischung von nebenläufigen Aktivitätsträgern

Maßnahmen

- Verhinderung von Umschaltungen durch Scheduler
  →Verknüpfung des Schedulers durch Koordinierung
- **Mutex** (Gegenseitiger Ausschluss): Nur ein Prozess / Thread in kritischem Abschnitt

4.10.4 Algorithmus von Peterson

Gegenseitiger Ausschluss zweier nebenläufiger Threads durch Variablen (ready und turn) durch aktives Warten
Problem Nur alternierendes Betreten des kritischen Abschnitts möglich, Verklemmung (*Live-lock*) möglich: kein Fortschritt beider Prozesse, aktives Warten (*Spin-lock*) 4.10.6

4.10.5 Maschinenbefehle für Mutex

- **TAS** (Test-and-Set): Atomare Sperre setzen →Wert einer Speicherzelle wird gelesen, auf einen "Lock"-Wert gesetzt (häufig 1) und der alte Wert zurückgegeben
- **SWP** (Swap): Atomarer Tausch zweier Speicherzellen →Speicherwert lesen, neuen Wert speichern und alten Wert ins Register zurückgeben
- **SEI** (Set-Interrupt-Flag): Sperrt Unterbrechungen (Interrupts)
- **CLI** (Clear-Interrupt-Flag): gibt Unterbrechungen frei

4.10.6 Spin-locks

Sperre zum Schutz gemeinsam genutzter Ressourcen, welche mittels aktivem Warten durch eine *Sperrvariable* umgesetzt wird
Vorteil: effizient für kurze kritische Abschnitte
Nachteil: Verbrauch von Rechenzeit ohne Nutzen, Behinderung anderer Prozesse

4.10.7 Semaphore

Systemdatenstruktur nach Dijkstra

- Semaphor verwaltet eine interne Zählervariable, welche initial auf den Wert der maximalen Anzahl an Prozessen die gleichzeitig auf eine durch den Semaphor geschützte Ressource zugreifen dürfen gesetzt wird
- Mit **P-Operation** kann der Zugriff auf die Ressource angefragt werden. Ist der Wert der Variablen größer Null, kann auf die Ressource zugegriffen werden und der Wert der Variablen wird um Eins verringert. Falls der Wert der Variablen kleiner/gleich Null ist, wird der anfragende Prozess in eine Warteschlange eingereiht und blockiert
- Mit **V-Operation** wird der Zugriff auf die Ressource wieder freigegeben und der Wert der Variablen wird wieder um Eins erhöht.
- P- und V-Operationen sind selbst *atomar*, um *Race Conditions* zu vermeiden

P-Operation (proberen): Wartet bis Zugang zum kritischen Abschnitt frei

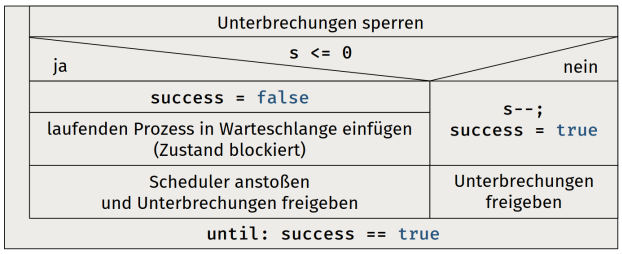

Figure 6: P-Operation

V-Operation (vrijgeven): Zugang für kritischen Abschnitt frei geben

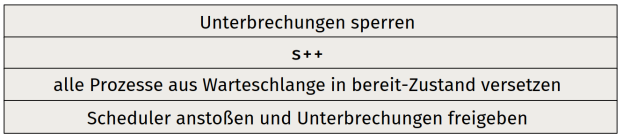

Figure 7: V-Operation

5 F Dateiverwaltung

5.1 Einheiten für Speicherkapazität

Unterscheidung KB/MB/GB (-lo) etc. ↔ KiB/MiB/GiB (-bi) etc.
Kilo/- Mega/- Giga/- ... byte: 10er Potenzen
Kibi/- Mibi/- Gibi/- ... byte: 2er Potenzen

5.2 Festplatten

Nicht-volatile (Non-volatile) Hintergrundspeicher

5.2.1 Hard Disk Drive (HDD)

Bestandteile Speicherplatte (Platter), Schreib/- Lesekopf (Head), Spuren (Tracks), Zylinder (Cylinder), Sektoren (Sectors)
Mikrocontroller organisiert mechanische Ansteuerung, und Blockzuteilung über SATA (Serial AT Attachment)
Identifikation eines Blocks durch **CHS-Information** (Cylinder, Head, Sector), kann Vielfaches der Sektorgröße sein (typischerweise 512B / 4 KiB)

5.2.2 Solid State Disk (SSD)

Nicht-mechanisch, Speicherung von Daten in Flash-Zellen
Zuordnung von physischen Blöcken zu einem logischen Block
Beim Löschen von Blöcken werden veraltete als ungültig markiert

5.3 Dateiverwaltung

Datei: speichert Daten als Folge von Bytes, Zugriff erfolgt sowohl **sequenziell** (Auslesen von Beginn bis Ende) als auch **wahlfrei** (Position in der Datei wählbar), dynamisch erweiterbar
Verzeichnis: Baumförmige Struktur, Knoten stellen Verzeichnisse dar, Blätter Verweise auf Dateien
Weitere Datenträger werden durch Systemaufruf mount in einen bestehenden Dateibaum montiert

5.3.1 Inodes

- Jede Datei bzw. jedes Verzeichnis besitzt Inode-Nummer, die Metadaten speichert

→Typ, Dateigröße, Eigentümer / Berechtigungen, Zeitstempel, Anzahl der Hard Links, Verweis auf Daten im Speicher

- **Direkt**: Nummer eines Blocks mit Dateiinhalt
- **Indirekt**: Verweis auf weitere Nummern von Blöcken (*Indirektionsstufen*)

Vorteile von indirekten Verweisen: sehr große Dateien adressierbar, schnellere Positionierung des Zeigers als z.B. bei der File Allocation Table bei FAT32, jedoch müssen Indirektionsblöcke zusätzlich geladen werden

5.3.2 Hard Links

- Zeigt auf Inode einer Datei
- Mehrere Verweise pro Datei möglich, diese wird erst dann gelöscht, wenn kein Hard Link mehr auf die Datei verweist
- Nicht für Verzeichnisse anlegbar, um Schleifen im Dateibaum zu vermeiden
- Verweise nur möglich innerhalb gleicher Dateisysteminstanz

5.3.3 Symbolic Links

- Speichert relativen Pfad zu Datei / Verzeichnis
- Nach Löschen der Datei bleibt der Link bestehen (Broken Link)
- Verweise über Dateisysteminstanz hinweg möglich

5.3.4 Systemaufrufe für Dateien

- open: Öffnen einer Datei, Zuweisung eines Filedeskriptor
- close: Schließen einer offenen Datei
- read: Lesen von einer Datei, verschiebt Zeiger um die Anzahl an Bytes, die gelesen wurden
- write Schreiben in Datei
- lseek: Positionieren des Zeigers in der Datei
  - SEEK_SET: absolute positionierung

- SEEK_CUR: relative Positionierung
- SEEK_END: relative Positionierung zum Dateiende

5.3.5 Systemaufrufe für Verzeichnisse

- **Lesen** *opendir, *readdir, closedir
- **Schreiben** open, link, symlink, unlink, mkdir, rmdir

5.3.6 Systemaufrufe für Inodes

- **Lesen** fstat, stat
- **Schreiben** utime, ...

5.4 Dateisysteme

Abstraktion zur einheitlichen Nutzung von Dateien auf Datenträgern: Nutzer muss sich nicht um Ansteuerung verschiedener Datenträger kümmern

- FAT32 (File Allocation Table), exFAT, NTFS (Windows), EXT4 (Linux), etc.
- **Partionierung** mehrere Dateisysteme pro Datenträger
- **Striping** Dateisystem über mehrere Datenträger (z.B. RAID 0)

5.4.1 FAT-Dateisystem (File Allocation Table)

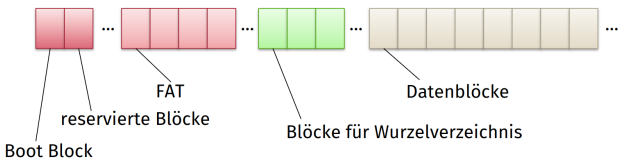

Figure 8: Blockorganisation unter FAT

- **Boot Block**: Informationen zum Mounten des Dateisystems, ggf. Bootloader zum Starten eines Betriebssystems
- **Reserviert**: Weitere Blöcke zum Booten, Sicherungskopie des Boot Blocks

- **FAT**: Speicherung von Datenblöcken einer Datei als verkettete Liste
- **Redundanz**: mehrfache Speicherung der FAT für Fehlertoleranz
- **Datenblöcke** speichern Dateien und Unterverzeichnisse
- **Blöcke** (auch Cluster genannt) 32 Bit groß, jedoch nur 28 Bit verwendet
  - frei = 0x00000000
  - belegt = Nummer des nächsten Block oder 0x0ffffff8-0x0fffffff für letzten Block
  - beschädigt = 0x0ffffff7

FAT Verzeichniseintrag

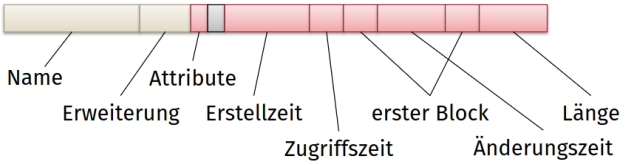

Figure 9: Verzeichniseintrag unter FAT

Attribute

- schreibgeschützt (read-only)
- versteckt (hidden)
- Systemdatei (system)
- Volume-Label (Bezeichnung für Datenträger)
- Unterverzeichnis (sub-directory)
- Archivbit (archive)
- Zeitstempel

Probleme unter FAT

- Zusätzliches Laden von mindestens einem Block
- Fragmentierung durch verteilte Cluster, da Daten an nächstmöglicher freier Stelle platziert wird
- Aufwändige Suche nach Datenblock bei bekannter Position in Datei

5.4.2 NTFS (New Technology File System)

Journaling-Dateisystem, speichert alle Daten, inkl. dem Dateisystem selbst, als Datei

Master File Table (MFT)

speichert Datei/- und Verzeichniseinträge mit Attributen als eigene *Datenströme*:

- Dateiname
- Zugriffsrechte (Access Control Lists / ACLs)
- Zeitstempel
- Dateigröße
- Dateninhalt / Speicherort der Daten

Bei sehr kleinen Dateien (<1KB) wird der Inhalt direkt im MFT gespeichert, um nicht zusätzlich Datenblöcke zu lesen

- **Cluster**: Speicherung von Daten in Clustern (512B bis 4KiB) mit logischer Cluster-Nummer als physische bzw. logische Adresse (LCN) und virtueller Cluster-Nummer (VCN) als Adresse innerhalb einer Datei
- **Extent**: aufeinanderfolgende Cluster werden als Extent gespeichert

NTFS File Reference

- **Sequenznummer**: Wird beim Verändern einer Datei inkrementiert, um veraltete oder ungültige Verweise zu erkennen
- **Dateinummer**: Eindeutiger Bezeichner für Datei / Verzeichnis

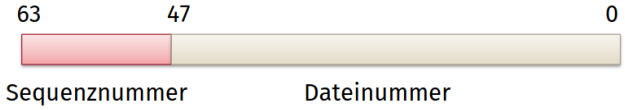

Figure 10: File Reference unter NTFS

Dateieinträge im MFT

Kleine Datei: Standard-Info, Dateiname, Daten passen direkt in den MFT Eintrag

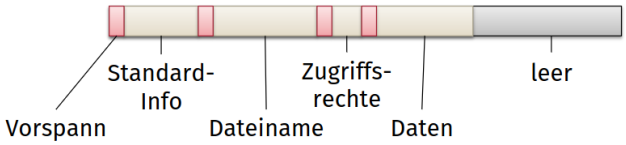

Figure 11: MFT Eintrag mit einer kleinen Datei

Große Datei: Erweiterung des Datenstroms durch Extent

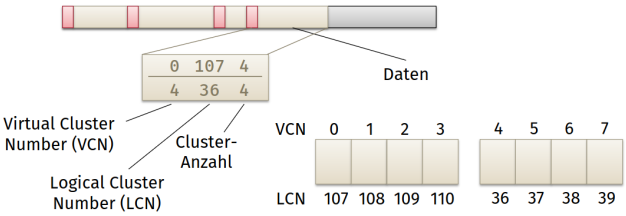

Figure 12: MFT Eintrag mit einer großen Datei

Kurzes Verzeichnis

- Verweis mit File Reference 5.4.2
- Name und Länge wird im Verzeichnis und in der Datei gespeichert →schnellerer Zugriff beim Listen eines Verzeichnisses

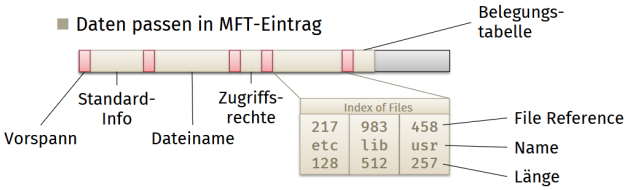

Figure 13: MFT Eintrag mit einem kurzen Verzeichnis

Langes Verzeichnis: Daten passen nicht in MFT Eintrag

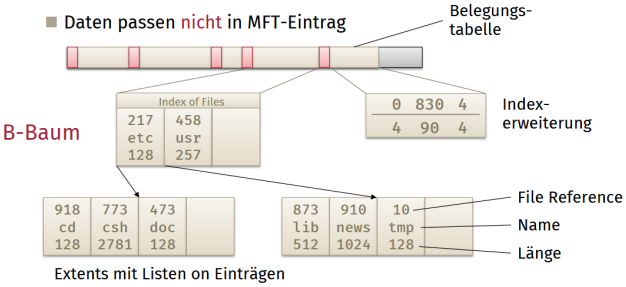

Figure 14: MFT Eintrag mit einem großen Verzeichnis

NTFS Dateiorganisation

Figure 15: NTFS Dateiorganisation

5.4.3 Journaling File System

- Protokollierung jeder Änderung (Transaktion)→Änderungen können wiederhergestellt (*Redo*) oder rückgängig (*Undo*) gemacht werden
- Kürzere Prüf-/ Reparaturphase
- NTFS, EXT3, ReiserFS

5.4.4 Log-structured File System

- Daten und Metadaten werden ans Ende eines Logs geschrieben, keine Überschreibungen
- Jeder Schreibvorgang erzeugt neue Datei, alte Dateien müssen durch Garbage-Collection aufgeräumt werden
- Robust gegenüber Systemabstürzen
- LinLogFS, BSD LFS, AIX XFS

6 G Speicherverwaltung

6.1 Logischer bzw. Virtueller Adressraum

- Abbildung von virtuellem Adressen auf Logische (physikalische Adressen) durch Hardware auf Seitenbasis (Seitenaddressierung)

- Möglicher Adressraum größer als realer Speicher →Auslagerung auf Swap möglich

6.2 Speichervergabe

Belegung durch

- Benutzerprogramme (Programmbefehle, Programmdaten)
- Betriebssystem (Betriebssystemcode, Pufferspeicher, Systemvariablen)

6.2.1 Statisch

Fester Bereich für Betriebssystem und Prozesse **Problem**: feste Zuteilung erforderlich

6.2.2 Dynamisch

- Einteilung in **Segmente**: zusammenhängender Speicherbereich
- Anforderung von Speicher bei Bedarf (Allokation), Freigabe nicht mehr benötigter Segmente →Zuteilungsalgorithmus
- **Code**-Segment: Instruktionen des Prozesses, Initialisierung durch Programmdatei
- **Daten**-Segment: vorbelegte Datenstrukturen (Programmdatei), andere Datenstrukturen (Nullen)
- **Stack**-Segment: Verwaltungsinformationen (verschachtelte Funktionsaufrufe, Unterbrechungen)

6.3 Dynamische Datenstrukturen

6.3.1 Array

- Fest definierter Speicherbereich
- Benötigt gleich viele aufeinanderfolgende Speicherblöcke wie Datenblöcke

6.3.2 Verkettete Liste

- Erster Block speicher Daten
- Zweiter Block speichert Referenz auf ersten Block des nächsten Elements
- FF_{16} markiert Ende der Liste

6.3.3 Verkettete Array-Liste

- Verkettung von Array-Blöcken variabler Größe
- Erster Speicherblock gibt Größe des gesamten Array-Blocks an
- Zweiter Speicherblock gibt Referenz auf nächsten Array-Block, FF_{16} markiert Ende
- Beliebige Anzahl an Datenblöcken pro Array-Block

6.4 Freispeicherverwaltung

6.4.1 Bitvektoren

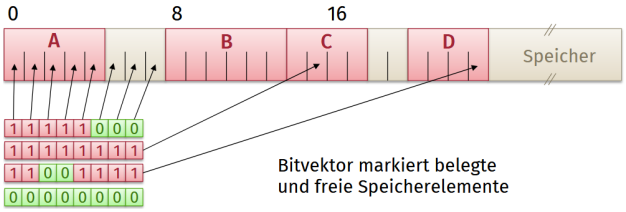

Figure 16: Freispeicherverwaltung mit Bitvektoren

6.4.2 Verkettete Liste

Repräsentation von freien und belegten Segmenten

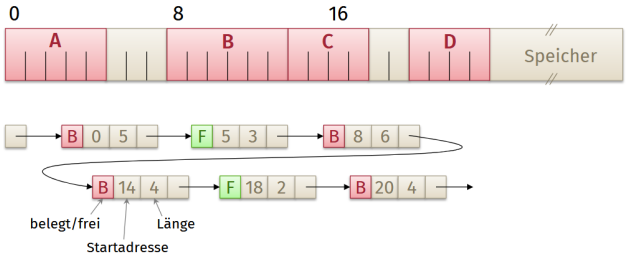

Figure 17: Freispeicherverwaltung mit Verketteter Liste

6.4.3 Verkettete Liste im freien Speicher

Repräsentation nur von freien Segmenten, Rückwärtsverkettung zur Effizienzsteigerung

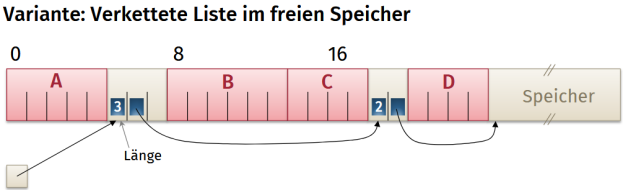

Figure 18: Freispeicherverwaltung mit Verketteter Liste im freien Speicher

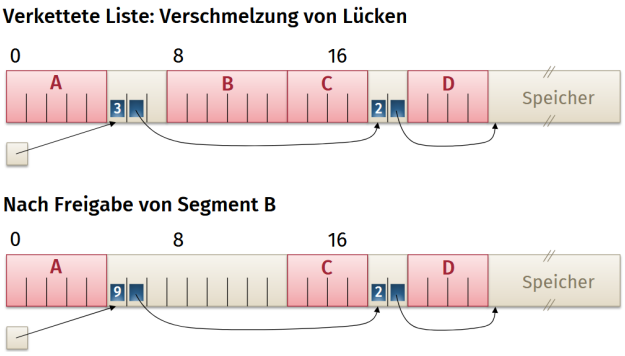

Figure 19: Verschmelzung von Lücken nach Freigabe eines Segments

6.4.4 Vergabestrategien

- **First Fit**: erste passende Lücke wird verwendet
- **Rotating First Fit / Next Fit**: wie First Fit, aber Start bei zuletzt zugewiesener Lücke
- **Best Fit**: kleinste Passende Lücke
- **Worst Fit**: größte passende Lücke wird gesucht

Probleme: Fragmentierung, Laufzeit des Algorithmus

Einsatz der Verfahren

- Betriebssystem (Systemspeicher, Zuteilung an Prozesse und Betriebssystem)

- Prozess (Verwaltung des Heap): dynamische Allokation durch Prozess
- Sekundärspeicher: Prozessauslagerungen (Swap)

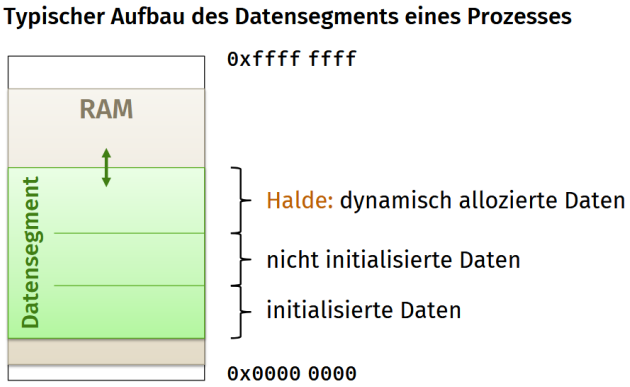

Figure 20: Datensegment eines Prozesses

6.5 Segmentadressierung

6.5.1 Physischer Adressraum

- Besteht aus DRAM, (VRAM), Boot-ROM, I/O
- nicht alle physikalischen Adressen verfügbar

Probleme unterschiedliche Adressen für Segmente z.B. für Prozesse des selben Programms, Hauptspeicher nicht ausreichend, keine Schutzumgebung

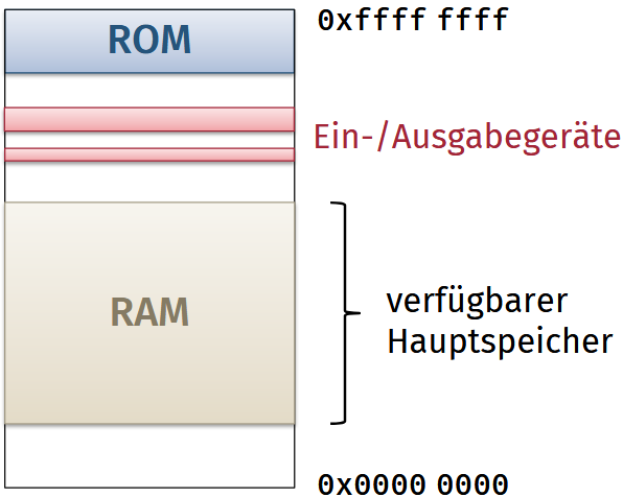

Figure 21: Physikalischer Adressraum

6.5.2 Logischer Adressraum

- Abbildung auf physischen Adressraum
- Jeder Prozess hat 2^{32} Adressen, jedoch evtl. nicht alle physisch vorhanden
- Programme können beliebige Adressen verwenden, ohne Kollisionen im Mehrprogrammbetrieb zu verursachen
- Nicht benötigte Segmente werden ausgelagert
- Schutzumgebung

→Hardware-Unterstützung erforderlich **Memory-Management-Unit** (MMU)

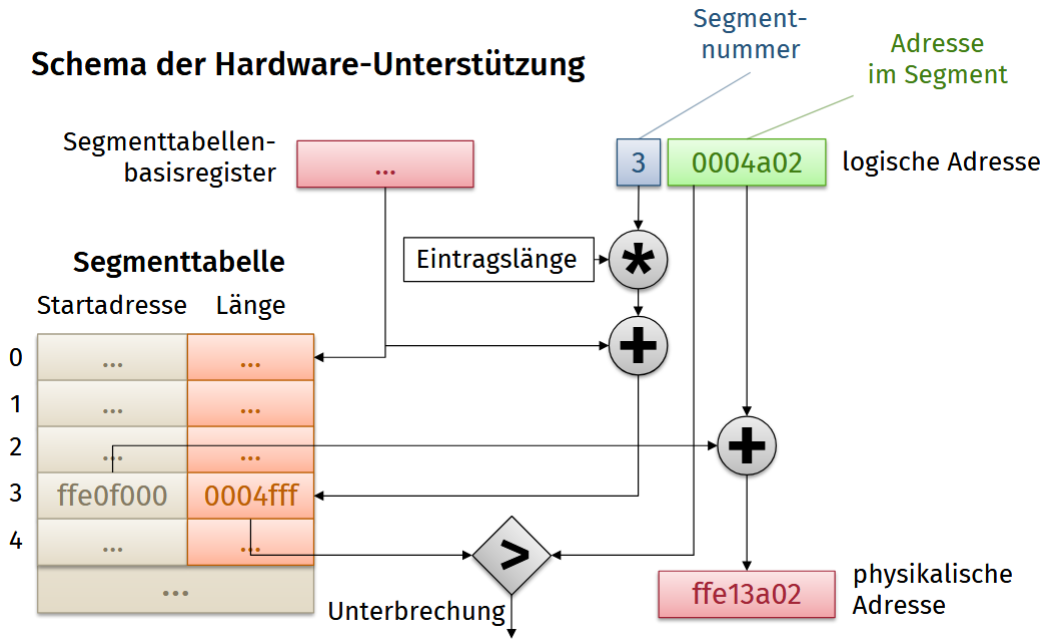

Figure 22: Segmentverwaltung durch MMU

6.5.3 Ablauf einer Adressübersetzung

Segmentnummer	Startadresse	Länge
00	FF00 F000	00 4FFF
01	FF11 F000	00 1FFF

Table 1: Beispielhafte Segmenttabelle

1. Extraktion der Segmentnummer, Offset innerhalb des Segments
2. Lesen des passenden Eintrags aus der Segmenttabelle (Startadresse + Länge des Segments)
3. Gültigkeitsprüfung der Adresse
4. Berechnung der physikalischen Adresse (Startadresse + Offset)

6.6 Seitenadressierung (Paging)

Seitennummer (VPN)	Präsenz-Bit	Startadresse
0	1	0000
1	0	2000
2	1	1000

Table 2: Beispielhafte Seitentabelle

- Einteilung von Segmenten in gleichgroße Seiten
- Ein/- Auslagerung von Seiten, nicht ganzen Segmenten
- *Präsenzbit* pro Seite speichert, ob Seite ein/- ausgelagert ist
- Beim Zugriff auf eine nicht-eingelagerte Seite entsteht ein *Page-Fault*

Vorteile: vereinfachte Speichervergabe und Auslagerung, kein Kompaktieren notwendig
Nachteile: Seitentabelle lang →mehr implizite Speicherzugriffe, Ungenützter Speicherplatz durch nicht vollständig gefüllte Seiten

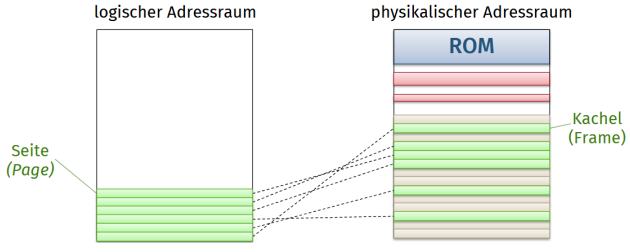

Figure 23: Abbildung von Seiten in physikalischen Adressraum

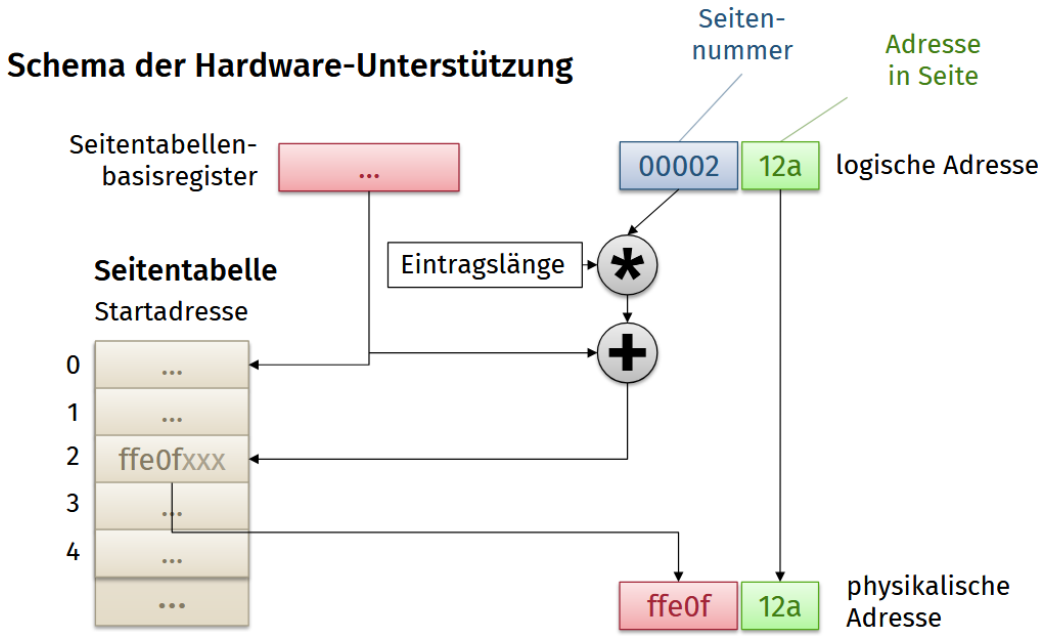

Figure 24: Seitenverwaltung (Paging) durch MMU

6.6.1 Ein- und Auslagerung

Einlagerung (Paging-In): Benötigte Seite ist nicht im Hauptspeicher (Unterbrechung Page-Fault), muss nachgeladen werden, andere Seite muss verdrängt werden

Auslagerung (Paging-Out): Hauptspeicher ist voll, daher müssen Seiten auf Festplatte gespeichert werden (z.B. LRU)

→Hardware-Unterstützung, Repräsenstation je nach Ein- bzw Auslagerug durch Präsenzbit

6.6.2 Seitenersetzung

Ersetzungsstrategien

- **First-in First-out**: Längster Block, der im Cache ist, wird entfernt
  - **Vorteile**: Einfache implementierung
  - **Nachteile**: Nichtbeachtung von Zugriffsverhalten, oft genutzte Daten können früh aussortiert werden

  - **Belady's Anomalie** (1969): größerer Hauptspeicher kann zu mehr Einlagerungen führen

- **Optimale Strategie**: In Realität nicht umsetzbar, da Seitenzugriffe vorhergesagt werden müssten
  - Seite, die am längsten nicht verwendet werden wird, wird verdrängt
  - Kontrollzustände sind die Vorwärtsabstände der Seiten
- **Least Recently Used**: Letzt verwendeter Datenblock wird entfernt
  - **Vorteile**: Berücksichtigung zeitlicher Lokalität
  - **Nachteile**: Komplex in HW-Umsetzung, hoher Verwaltungsaufwand

  Second Chance, Clock: Annäherung an LRU, verwendet Referenzbit (Used Bit), das beim Verwenden auf 1 und nach Wartezeit auf 0 gesetzt wird, Seiten mit dem Referenzbit 0 werden dann ersetzt
  - **Vorteile**: Geringer Speicherbedarf, einfache Implementierung
  - **Nachteile**: Bei vielen Seiten hat Zeiger nicht genug Zeit, Referenz-Bits auf 0 zu setzen →Degeneration zu FIFO
  - **Variante**: zusätzliche Berücksichtigung eines Modifikationsbit (Dirty Bit / DB)
  - Anpassung der Zeigergeschwindigkeit möglich

6.6.3 Hierarchische Seitenadressierung

- Verwaltung großer virtueller Adressräume
- Einteilung in mehrstufige Page Tables
- Jede Ebene interpretiert eine Teilmenge der Adressbits
- Einträge in einem *Page Directory* verweisen auf untergeordnete *Page Tables*, die schließlich die physische Seitenadresse enthalten
- Reduzierung des Speicherbedarfs, da Tabellen nur bei Bedarf angelegt werden

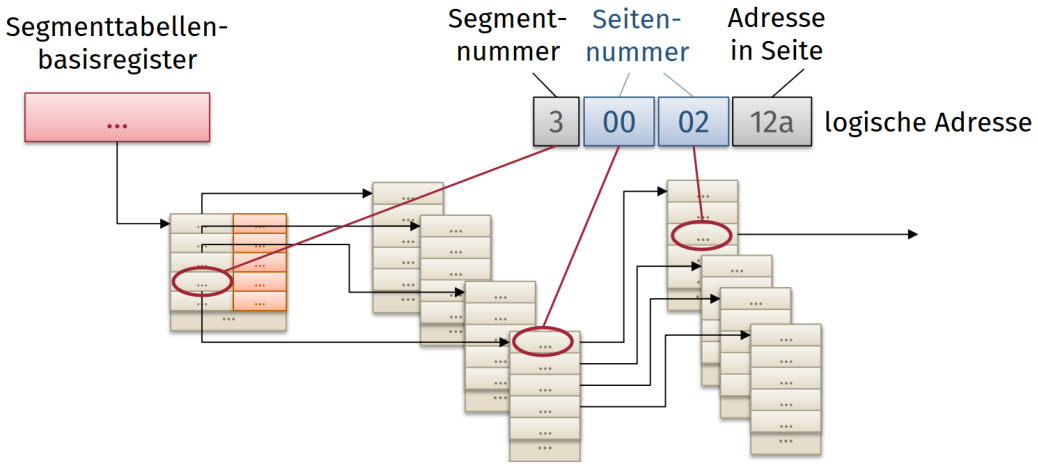

Figure 25: Begrenzung der Seitentabellen durch Indirektionsstufen

6.6.4 Demand Paging

- **Seitenfehler (Page Fault)**: Eine Unterbrechung tritt auf, weil auf eine noch nicht geladene Seite zugegriffen wird.
- **Seitenverdrängung**, falls kein freier Seitenrahmen verfügbar ist
- **Einlagerung der benötigten Seite** aus dem Sekundärspeicher
- **Wiederholung des unterbrochenen Zugriffs**, nachdem die Seite erfolgreich geladen wurde.

6.6.5 Freiseitenpuffer

- Vom Betriebssystem verwaltete Datenstruktur, die freie Seitenrahmen im Hauptspeicher speichert
- Ermöglicht schnelle Zuteilung von Speicher bei Seitenfehlern oder Speicheranforderungen, ohne sofort Seiten ersetzen zu müssen
- Ist Puffer erschöpft, werden Seiten präventiv ausgelagert, um ihn wieder aufzufüllen

6.6.6 Seitenflattern (Trashing)

Betriebssystem verbringt so viel Zeit mit dem Ein- und Auslagern von Seiten (Paging), dass kaum noch Zeit für die eigentliche Programmausführung bleibt.
Ursachen

- Prozess nahe am *Seitenminimum* (minimale Speicherbedarf, sodass der Prozess effizient laufen kann)
- Gleichzeitige Ausführung von zu vielen Prozessen
- Schlechte Ersetzungsstrategie

6.6.7 Mehrprozessbetrieb

Ersetzungsstrategien

Lokal: Statische Zuteilung von Kacheln pro Prozess, erfordert Zuordnung einer Kachelmenge zu Prozess

Varianten

- *Statisch*: Hauptspeicher / Anzahl Prozesse, proportional zu Prozessgröße z.B. nach Segmentgrößen
- *Dynamisch*: Proportional zum Kachelbedarf →Working-Set-Ansatz

Global: Prozess ersetzt Seiten anderer Prozesse →dynamisches Verhalten der Prozesse, ungenutzte Kacheln können von anderen Prozessen verwendet werden

6.6.8 Deaktivieren von Prozessen

Inaktiver Prozess wird vollständig ausgelagert (swapped out)

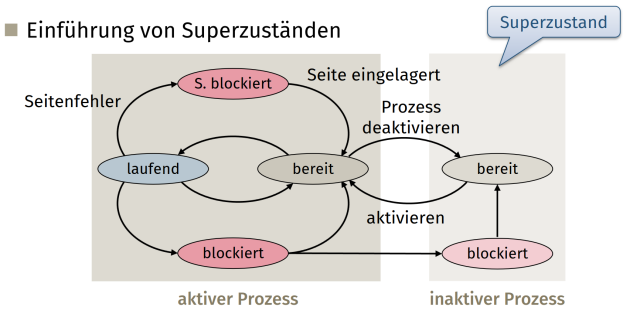

Figure 26: Deaktivieren von Prozessen durch Superzustände

6.6.9 Systemaufrufe

Segmentgrößen

- *Code-Segment*: statische Größe
- *Daten-Segment*:
  - int err = brk(void *endaddr);
  - void *newend = sbrk(intptr_t increment);
  - dynamisches Anpassen des Segmentendes (Program break)
- *Stack-Segment*: dynamisch und automatisch angepasst

Einblenden von Dateien
Abbildung von Dateien in den Hauptspeicher

```
void *newaddr = mmap( void *addr, size\_t length, int prot,
int flags, int fd, off\_t offset );
```

- addr: Startadresse im logischen Adressraum
- length: Länge des Mappings
- prot/flags: Konfigurationsangaben
- fd: geöffnete Datei
- offset: ab welcher Stelle in der Datei soll eingeblendet werden

Ausblenden

```
int errno = munmap( void *addr, size_t length );
```

Gemeinsamer Speicher (Shared-Memory-Object)
Filedeskriptor (fd) kann mit mmap() verwendet werden

```
int fd = shm_open( const char *name, int oflag, mode_t mode);
```

- *name*: globaler Name für das Objekt
- *oflag*: ähnliche Bedeutung wie bei open()
- *mode*: Berechtigungen

Ändern der Länge

```
int errno = ftruncate( int fd, off_t length );
```

Löschen des Shared.Memory-Objects

```
int errno = shm_unlink( const char *name );
```

6.7 Memory-Management-Unit

6.7.1 Vorteile

- **Schutzumgebung**: Unterbrechung bei Speicherverletzung
- **Kontextwechsel**: zusätzlicher Austausch der Segmentbasis
- **Ein-/ Auslagerung**: Anpassung Segmenttabelle
- **Gemeinsame Segmente**: Code-Segmente, Datensegmente (Shared Memory)
- **Rechteintegration**: Segmenttabelleneintrag

6.7.2 Nachteile

- **Implizite Speicherzugriffe**: zusätzlicher Zugriff auf Segmenttabelle
- **Ein-/ Auslagerung** grobgranular: lange I/O-Zeiten
- **Fragmentierung**: nach Ein-/ Auslagerung eventuell nicht nutzbare Lücken →**Kompaktieren**: Verschieben der Segmente

6.8 Translation Look-Aside Buffer

Kleiner, schneller Cache, der bei der Adressumsetzung von virtuellen zu physischen Adressen verwendet wird. Er befindet sich typischerweise in der MMU (Memory Management Unit) der CPU.

- **TLB Hit**: Adressübersetzung bereits in TLB →Direkter Zugriff auf physische Adresse
- **TLB Miss**: Übersetzung nicht im TLB, Durchsuchung der Seitentabelle erforderlich, Ersetzung eines alten Eintrags (z.B. LRU)
- **Flush** bei Kontextwechsel

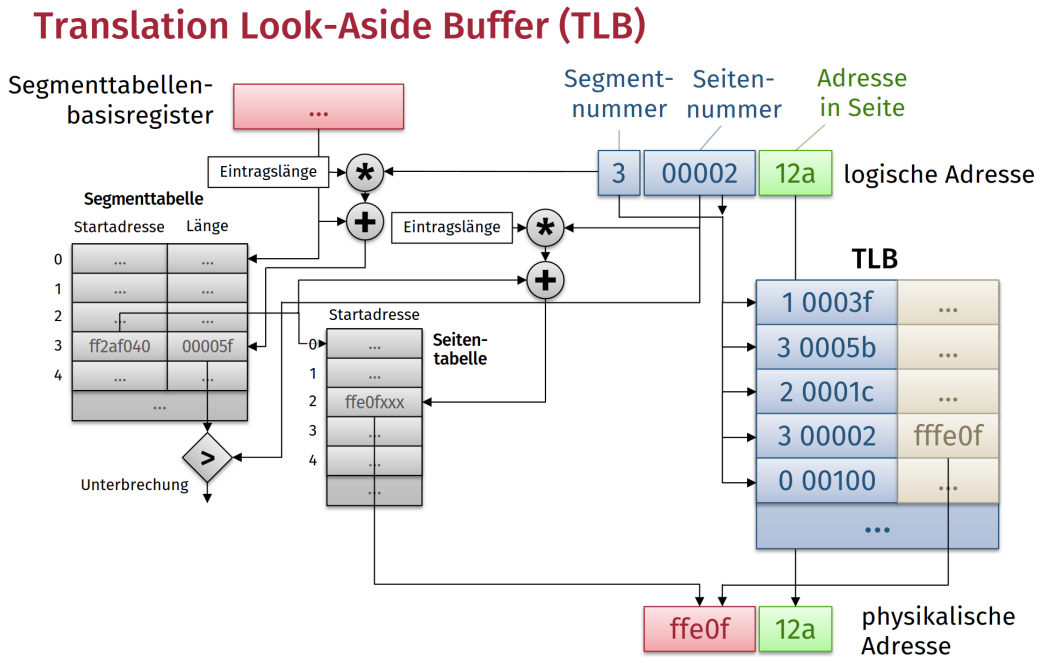

Figure 27: Translation Look-Aside Buffer

7 H Input / Output (IO)

7.1 Aufgaben eines Treibers

- Verwaltung von Zugriffsrechten
- Optimierung der Gerätenutzung
- Zuteilung von Ressourcen an Benutzerprozesse
- Auflösung von Zugriffskonflikten
- Datentransport von und zum Gerät

7.2 Treiberimplementierung

- *Synchrone* IO: Aufruf Treiber wartet, bis I/O-Operation abgeschlossen ist (Bsp.: Ablauf eines synchronen Leseaufrufs)
- *Asynchrone* IO: I/O-Operation wird angestoßen, aber der aufrufende Thread wird nicht blockiert. Nachträgliche Synchronisation beim Abschluss evtl. erforderlich

7.2.1 Polling-Betrieb

- Prozessor einzige aktive Instanz bei I/O-Operation
- *Aktives Warten*, bis Gerät bereit ist, dann eigentliche Datenübertragung

Vorteile: Einfache Implementierung
Nachteile: *Aktives Warten* verschwendet Rechenzeit, *Implizites Blockieren* anderer Datenträger

7.2.2 Unterbrechungsbetrieb

- Gerät löst Unterbrechung (3.5.1) aus
- Prozessor startet Unterbrechungsbehandlung

Bei mehreren I/O-Operationen

- Interne Warteschlange
- Blockierung des Aktivitätsträger (synchrone I/O)
- Strategische Abarbeitung der Aufträge (z.B. FIFO, Prioritätsbasiert)
- Freigabe blockierter Aktivitätsträger innerhalb ISR
- Synchronisierung bei asynchroner I/O-Operation

Vorteile: Prozessor muss nicht warten, kann währenddessen andere Instruktionen ausführen
Nachteile: Komplexes Programmiermodell, Zusatzwaufwand durch Unterbrechungsbehandlung

7.2.3 Plattentreiber

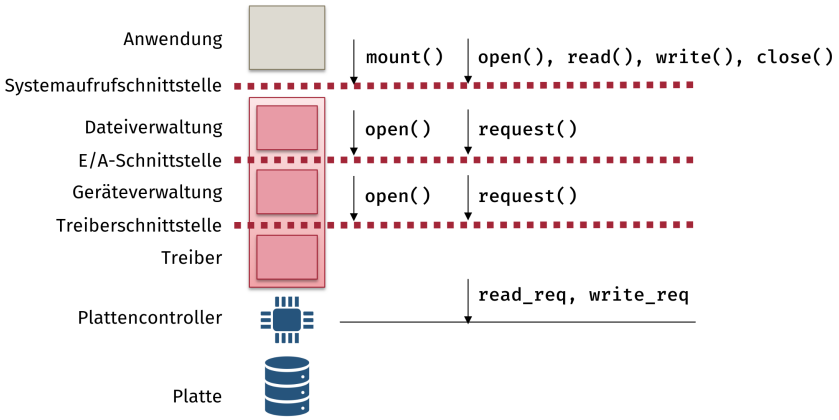

Figure 28: Software und Hardware zwischen Anwender und Platte

Ablauf eines lesenden Systemaufrufs

- Überprüfung, ob Block im Cache vorhanden, falls nicht: *request()* im entsprechenden Treiber
- Ausführung von *request()* stößt Plattenpositionierung an
- Blockierung des Prozesses, Scheduler führt andere Prozesse aus
- Plattencontroller signalisiert erfolgreiche Positionierung, ISR führt Prozess fort, stößt Sektorsuche an
- Plattencontroller meldet Abschluss des Durchlaufs eines Sektors, ISR liest Daten im Pollingbetrieb
- Deblockierung des Prozesses

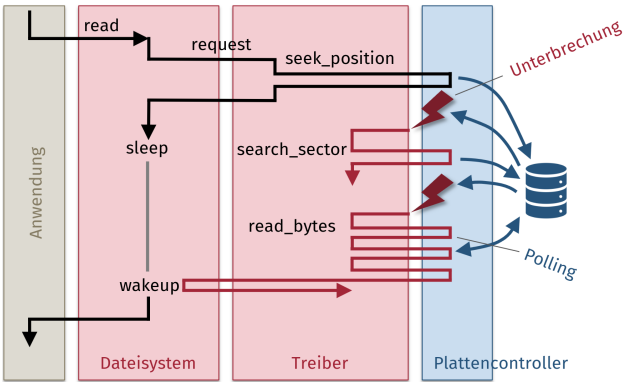

Figure 29: Ablauf eines synchronen Leseaufrufs

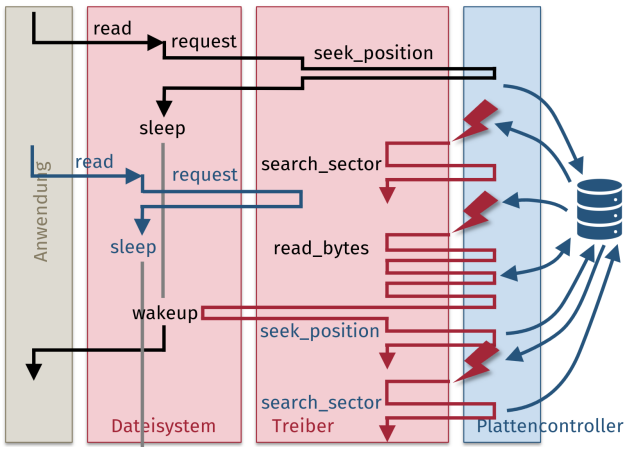

Figure 30: Ablauf mehrerer synchroner Leseaufrufe

7.3 Direct Memory Access (DMA)

Lesen von Speicher und Weitergeben der Daten an I/O-Baustein und umgekehrt durch CPU erzeugt hohe Wartezeiten besonders bei großen Datenmengen wie z.B. bei Festplatten
DMA ist Aktor auf Speicherbus, Entlastung der CPU
Programmierung: Länge, Speicheradresse, I/O-Adresse von I/O-Gerät
Arbeitsmodi

7.3.1 Burst-Modus

DMA-Baustein übernimmt Systembus für die Dauer des vollständigen Transfers

Vorteil: Hohe Transferrate
Nachteile: CPU für lange Zeit am Speicher-/Buszugriff gehindert

7.3.2 Cycle Stealing

Durchmischen von DMA- und CPU-Buszyklen
DMA nutzt alle von CPU nicht benötigten Buszyklen
Vorteil: Geringfügige Einschränkung der CPU-Auslastung
Nachteil: Geringere Transferrate

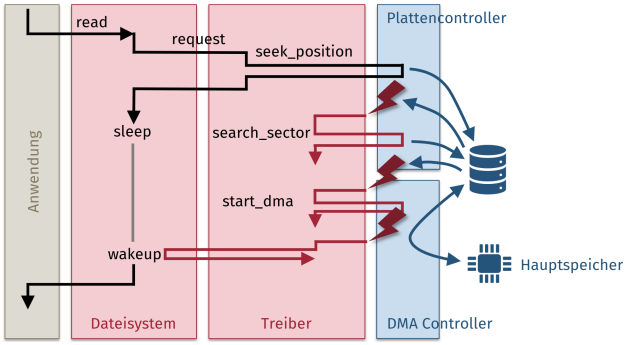

Figure 31: Ablauf eines synchronen Leseaufrufs mit DMA

7.3.3 Treiber mit DMA

Große Systeme mit mehreren DMA-Kanälen & vielen Platten

- Suche nach freiem DMA-Kanal, Blockierung
- Anforderung des DMA-Kanals kann parallel zur Plattenpositionierung erfolgen
-

Mainframe-Systeme

- Steuereinheit fasst mehrere Platten zu einem Gerät zusammen
- Steuereinheiten hängen an einem Kanal zum Hauptspeicher
- *Teilwegbelegung*: erst Belegung der Steuereinheit, dann des Kanals beim Zugriff auf Platten

7.3.4 Caching

- DMA-Controller greift direkt auf RAM zu, ohne Cache zu berücksichtigen
- Verhinderung von Dateninkonsistenzen zwischen RAM und Cache
  - Write-Back des Cache: Daten aus Cache müssen in RAM geschrieben werden für DMA-Controller
  - Invalidierung des Cache: Von DMA modifizierte Daten müssen im Cache invalidiert werden

Bus-Mastering: Daten können direkt vom DMA zur Platte übertragen werden
z.B. bei USB-Tastaturen, Netzwerkkarten

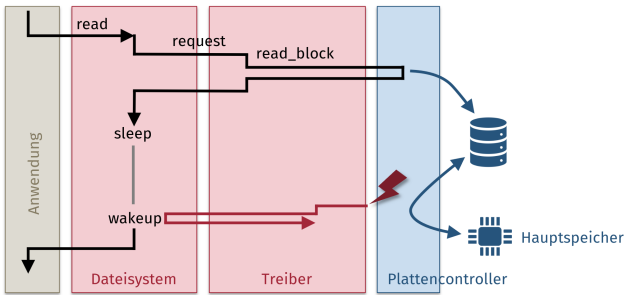

Figure 32: Plattencontroller als Bus-Master

7.4 Repräsentation von I/O-Geräten

Spezialdatei im Dateisystem, besteht aus Inode ohne Daten, diese speichert *Major* und *Minor* Number & Typ

- *Major* Number: Verweis auf Treiber
- *Minor* Number: Verweis auf vom Treiber verwaltetes Gerät

Typen:
Blockorientiert: Platten, CD-Rom, ...
Zeichenorientiert: USB-Geräte, Serielle Schnittstellen, Audiokanäle, ...
Blockorientierte Geräte häufig auch Zeichenorientiert repräsentiert

7.4.1 Blockorientierte Geräte

- Globaler Cache (Block Buffer Cache), Caching von Lese-/ Schreiboperationen
- Hauptspeicher: Verwaltung des Bezugs zu Gerät (*Major* Number, *Minor* Number, *Logische Blocknummer*)
- G Speicherverwaltung: Seitenverwaltung, Lesen / Schreiben von Blöcken, Einblenden von Dateien in logischen Adressraum

7.4.2 Serielle Schnittstellen

- Zeichenorientiert
- Neben *read() & write() ioctl()* (I/O-Control) Systemaufruf
- Treiberabhängige Parameter zur Konfiguration (Einstellungen, Editierfunktionen, Softwareflusskontrolle)

7.4.3 Unterscheidung im Verzeichnislisting

Beide Geräte-Arten können durch das Zeichen in der ersten Spalte des Listing-Outputs unterschieden werden

- Blockorientiert: b (block device)
- Zeichenorientiert: c (chracter device)

7.5 Disk-Scheduling

Strategische Abarbeitung der Warteschlange zur Effizienzsteigerung

- *Positionierzeit*: abhängig von Position des Magnetkopf der Festplatte
- *Latenzzeit*: Zeit, bis Magnetkopf Sektor erreicht
- *Übertragungszeit*: Zeit zur Übertragung der Daten

Scheduling-Strategien für Positionierzeit

7.5.1 First-Come, First-Served (FCFS)

Bearbeitung nach Eintreffen des Auftrags in der Warteschlange

7.5.2 Shortest Seek Time First (SSTF)

Tie-Break Heuristik bei gleicher Positionierzeit, d.h. bei gleicher *Seek-Time* von zwei Datenmengen wird z.B. nach niedrigster Blocknummer entschieden
Problem: *Aushungerung* möglich (wie z.B. bei Highest Priority First (HPF))

7.5.3 SCAN

Bewegen des Magnetkopfes in eine Richtung, bis keine Aufträge mehr in dieser Richtung vorhanden sind
Bei neuen Aufträgen Keine zusätzliche Positionierzeit, keine *Aushungerung*
C-SCAN (Circular SCAN): Bewegung nur in eine Richtung, Rückkehr zum Anfang

7.6 Windows Plug-and-Play (PnP)

Kompatible Geräte gekennzeichnet durch *Device ID*, *Hardware ID*, *Compatible IDs* (Kompatibilität mit anderen Geräten)
USB-Geräte

- Abfrage Hardware ID, standardisierte USB-Nachrichten
- *inf-Dateien* in der Registry: Suche nach passendem Treiber

7.7 Windows Power Management

Advanced Configuration and Power Interface (ACPI)

- Versetzen von Geräten bzw. des gesamten Systems in energiesparende Zustände
- Unterstützung von Mainboard und BIOS erforderlich
- **Systemzustände** *S0* bis *S5*
- **Gerätezustände** *D0* bis *D3*
- Betriebssystem verwaltet Systemzustände, Power-Manager benachrichtigt Treiber

7.7.1 ACPI-Systemzustände

- *S0 (Working)*: Normaler Betriebsmodus
- *S1 (Standby)*: Reduzierte Energieaufnahme durch Anhalten der CPU, jedoch Erhaltung des Kontext von CPU und Speicher
- *S2 (Standby)*: Energiesparender als S1, Kontext von CPU und Caches geht verloren und muss gesichert werden
- *S3 (Suspend-to-RAM, Standby, Sleep)*: CPU, Caches, Teile des Mainboard ausgeschaltet, Hauptspeicher erhält weiterhin Strom
- *S4 (Suspend-to-Disk, Hibernation)*: System vollständig abgeschaltet, Speicherinhalte und Kontexte werden auf Platte gespeichert
- *S5 (Off)*: System ausgeschaltet, kein Energiebedarf

7.7.2 ACPI-Gerätezustände

- *D0 (Fully-On)*: Gerät vollständig aktiv und einsatzbereit
- *D1*: Reduzierte Stromaufnahme, Kein Zugriff mehr durch Treiber
- *D2*: Weiter reduzierte Stromaufnahme zu D1
- *D3 (Off)*: Gerät vollständig von Stromversorgung getrennt, muss neu initialisiert werden

7.8 Zeichensätze

7.8.1 ASCII

- American Standard Code for Information Interchange
- 7 Bit Code
- Zeichen & Steuercodes

Probleme: Zeichen aus anderen Sprachen als Englisch nicht unterstützt

7.8.2 ISO-8859

- International Organization for Standardization
- 8 Bit Code
- ISO-8859-1 / Latin 1: Westeuropäischer Zeichensatz
- ISO-8859-15 / Latin 9: Westeuropäischer Zeichensatz mit Zeichen wie z.B. €

7.8.3 Unicode

- Einteilung von *Codepunkten* Code Units (Eindeutige Nummer für ein Zeichen) in *Planes* (Ebenen)
- 17 Planes mit jeweils 65.536 Zeichen (U+0000 bis U+10FFFF)
- Basic Multilingual Plane (BMP, Plane 0): alle aktuell gebräuchlichen Satzzeichen & Symbole
  - U+0000 bis U+007F identisch mit ASCII
  - U+0080 bis U+00FF identisch zu Latin-1
- Kontinuierliche Ergänzung um weitere Schriftzeichen
- Vereinbarung von Informationen wie Schreibrichtung, Normalisierung mehrdeutiger Darstellungen

7.8.4 UTF-8

UTF = Unicode Transformation Format

- 8-Bit Code, jedoch mehrere Codes für manche Zeichen erforderlich (bis zu 4 Byte)
- U+0000 bis U+007F identische Codierung wie ASCII
- U+0080 bis U+00FF selben Zeichen wie Latin-1, jedoch 2 Byte lang

Zeichenlänge

- 0xxxxxxx : 1 Byte
- 110xxxxx : 2 Byte
- 1110xxxx : 3 Byte

- 11110xxx : 4 Byte
- 10xxxxxx : gehört zu vorherigem Zeichen

Beispiele

- A = U+000041 = 41_{16}
- ä = U+0000E4 = $C3\ A4_{16}$
- € = U+0020AC = $E2\ 82\ AC_{16}$

7.8.5 UTF-16

- 16-Bit Code, jedoch mehrere Codes für ein Zeichen möglich (bis zu 4 Byte)
- U+0000 bis U+00FF selben Zeichen wie Latin-1, jedoch 2 Byte lang
- U+010000 – U+10FFFF (außerhalb der BMP Plane) wird in Surrogate Pairs (zwei 16 Bit Werte) aufgeteilt

Beispiele

- A = U+000041 = $00\ 41_{16}$
- ä = U+0000E4 = $00\ E4_{16}$

7.8.6 UTF-32

- 32-Bit Code
- U+0000 bis U+00FF selben Zeichen wie Latin-1, jedoch 4 Byte lang
- Hoher Speicherbedarf

Beispiele

- A = $00\ 00\ 00\ 41_{16}$
- ä = $00\ 00\ 00\ E4_{16}$

7.8.7 Vergleich von Unicode Codierungsformaten

- **UTF-32**: jedes Wort 32 Bit groß, wahlfreier Zugriff effizient
- **UTF-16**: 16-Bit Wort, dass mit Bitfolge 110111 beginnt zeigt zweite Hälfte an, d.h. vorheriges 16-Bit Wort ist Beginn des codierten Zeichens
- **UTF-8**: 8-Bit Wort, dass mit 10 beginnt, zeigt zweites, drittes oder viertes Byte an. Das erste vorherige Byte, dass dieses Muster nicht zeigt, ist Beginn des codierten Zeichens

8 I Virtualisierung

- Mehrere *virtuelle* Maschinen auf realer Maschine
- Isolation zwischen virtuellen Maschinen

8.1 Hardware-Partitionierung

- Physische Aufteilung der Hardware in mehrere Hardware-Partitionen, auf denen jeweils ein eigenes Betriebssystem läuft z.B. eigene CPU, RAM, I/O
- Kein gemeinsamer Kernel oder Hypervisor
- Dynamische Anpassung während des Betriebs möglich
- Mainframe-Systeme & Enterprise-Server

8.2 Virtual-Machine-Monitor (VMM)

Auch *Typ-2-Hypervisor* bzw. *Typ-2-VMM* genannt

- Virtuelle Maschine (VM) läuft als Anwendungsprozess im Wirtsbetriebssystem (Wirts-BS), enthält vollständige virtualisierte Hardware für Gast-Betriebssytem
- **Emulation** möglich: Abbildung physisch vorhandener Hardware
- Teilweise Emulation möglich: Hardwarebausteine, die nicht durch mehrere Gast-BS verwendbar sind
- VMWare Server, MS Virtual PC, VirtualBox

8.3 Paravirtualisierung

Auch *Typ-1-Hypervisor* oder *Typ-1-VMM* genannt

- **Hypervisor** als Vermittlungsschicht zwischen Hardware und virtuellen Betriebssystemen
- Hypervisor als Metabetriebssystem: nur APIs für Gast-Betriebssysteme
- Hypervisor-Aufruf statt direkter Hardware-Zugriffe, Zugriff erfolgt über spezielles System-Gastbetriebssystem
- Direkte Verwaltung von Hardware-Ressourcen durch Hypervisor

Vorteile: effizienter, da Optimierung für Virtualisierung, Vermeidung von Emulationen
Nachteile: Anpassung des Betriebssystem notwendig, z.B. eigene Kernmodule

8.4 Betriebssystemvirtualisierung

- Container: voneinander isolierte Anwendungsdomänen
- Docker: häufig verwendeter Container-Manager

Vorteile: effizienter als Paravirtualisierung, weniger Overhead
Nachteile: nur ein BS verwendbar, geringere Isolation

8.4.1 Beispiel Docker

- Erfordert unter Windows / MacOS Unterstützung eines VMM
- *Docker Image*: beinhaltet Software + Konfiguration, Erstellung durch eine Beschreibung (*Dockerfile*), Speicherung in *Docker Registry* (Verteilstelle für Images)
- Start eines Containers: Ermittlung des Images aus der *Docker Registry*, laden des Images, einmalige Ausführung eines Kommandos und Start des Dienstes
- Betrieb eines Containers: *Port-Forwarding*, virtuelle Netzwerke zwischen Containern
- Häufig Einsatz im Cloud-Computing, Cloud-Provider bietet Virtuelle Maschinen, auf denen Container ausgeführt werden →Doppelte Virtualisierung

8.5 Mechanismen für Virtualisierung

8.5.1 Unterbrechungen auf Prozessebene

Signalverarbeitung in Prozessen

- **SIGUSR1**: Prozesskommunikation
- **SIGINT**: Tasteneingabe
- **SIGSEGV**: Ungültiger Speicherzugriff
- **SIGALARM**: Timer
- **SIGILL**: Illegaler Maschinenbefehl

Anwendung in der Virtualisierung

- Hardware Emulation (z.B. Zugriff abfangen)
- Benutzerdefinierte Thread-Wechsel
- Emulation privilegierter Befehle

8.5.2 Prozessunterstützung für Virtualisierung

Problem: Gast-BS läuft mit Supervisor-Rechten (S=1), VMM läuft mit Nutzerrechten (S=0)
Lösung: Einführen eines zusätzlichen Modus: *Virtueller Modus*, Privilegierte Befehle lösen Unterbrechung aus, Hypervisor entscheidet, ob Befehl erlaubt ist

9 J Deadlocks

9.1 Philosophen Problem

- Philosoph kann entweder essen oder denken
- Zwischen zwei Philosophen liegt jeweils nur eine Gabel
- Philosoph nimmt zuerst linke, dann rechte Gabel
- **Problem**: Philosophen können verhungern

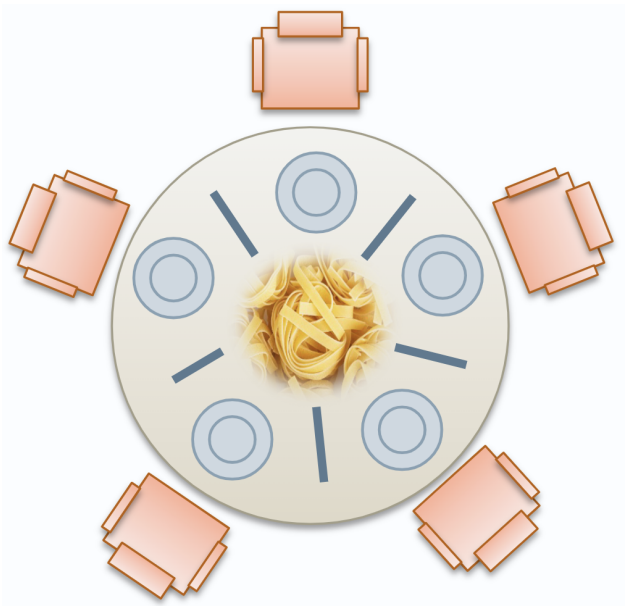

Figure 33: Abstraktionsebenen am Beispiel von Java

9.2 Livelock

- Prozesse laufen, machen jedoch keinen Fortschritt
- Gegenseitiges Ausweichen der Prozesse
- Ständige Zustandsänderung

9.3 Betriebsmittel / Ressourcen

- CPU, Drucker, Geräte
- Virtuelle Betriebsmittel der Anwendung bzw. des Betriebssystems

9.3.1 Unterscheidung Typ / Instanz

- Typ: eindeutige Definition eines Betriebsmittels
- Instanz: Ausprägung eines Typs z.B. mehrere CPUs

9.3.2 Betriebsnutzung

- Anfordern des Betriebsmittels (Belegung), Blockierung bei exklusiver Nutzung

- Nutzung des Betriebsmittels
- Freigeben des Betriebsmittels

9.4 Deadlock (Verklemmung)

9.4.1 Voraussetzungen für Verklemmungen

Vier notwendige Bedingungen

1. **Exklusive Belegung**
2. **Nachforderung** von Betriebsmittel (Prozess fordert weiteres Betriebsmittel an)
3. Kein Entzug von Betriebsmittel möglich (Betriebsmittel kann nicht zurückgefordert werden, bis sie der Prozess wieder freigibt)
4. **Zirkuläres Warten** (Ring von Prozessen, in dem jeder auf ein Betriebsmittel wartet, dass ein anderer verwendet)

9.4.2 Vermeidung von Verklemmungen

Vermeidung der notwendigen Bedingungen

1. **Exklusive Betriebsmittel**: in der Regel nicht verhinderbar
2. **Nachforderung von Betriebsmitteln**:
   - z.B. Gabeln in atomarer Belegung (beide oder keine)
   - Nachteile: Ungenutzte Betriebsmittel, Aushungerung möglich
3. **Kein Entzug von Betriebsmitteln**:
   - Gehaltene Betriebsmittel werden freigegeben und zusammen mit neuen angefordert
   - Während Warten werden bisher gehaltene frei für andere
   - Möglich für CPU / Speicher (Auslagerung), jedoch unmöglich für z.B. Drucker
4. **Zyklus**: totale Ordnung in der Betriebsmittel ausschließlich angefordert werden dürfen

Zyklusvermeidung: Fehlermeldung bei Verletzung der Ordnung, Rückgabe der Betriebsmittel, Anforderung in richtiger Reihenfolge
Problem: nicht immer realisierbar, da nächste Anforderung nicht bekannt und Rückgabe während Operation nicht möglich

9.4.3 Vermeidung durch Prävention

- **Sicherer Zustand**: System gerät nicht in Verklemmung
- **Unsicherer Zustand**: System wird sich verklemmen, läuft aber noch
- **Unmöglicher Zustand**: Verletzung der Semantik von Semaphoren z.B. doppeltes Aufrufen der P-Operation
- Erkennung durch z.B. Bankers Algorithmus

9.4.4 Erkennung von Verklemmungen

- Zykluserkennung durch Wartegraph
- Graphische Reduktionsmethode
- **Problem**: Erkennung rechenaufwendig, jedoch sind Deadlocks eher selten
- **Herangehensweisen**: Einsatz bei Blockierung, periodisch, oder wenn Auslastung sinkt

9.4.5 Erholung bei Verklemmungen

- Terminierung von Prozessen
- Verklemmten Prozesse abbrechen
- Einen Prozess abbrechen und Erkennung wiederholen
- **Probleme**
  - Verlust von berechneten Informationen
  - Rücksetzbarkeit von bisherigen Effekten

9.4.6 Einsatzgebiete von Antiverklemmungsmaßnahmen

- **Betriebssysteme**: Intere Vermeidung durch Ordnung
- **Anwendungsprozesse**: Keine Unterstützung durch Betriebssystem
- **Datenbanksysteme**: Rücksetzbarkeit von Datenbanktransaktionen unterstützt Erholung

10 K Rechte

10.1 Berechtigungen

10.1.1 Einschränkung des Zugriffs

- Mehrprozesssysteme: Prozesse
- Mehrbenutzersystem: Benutzer

10.1.2 Zugriffsmöglichkeiten

- Datenzentriert: lesender und schreibender Zugriff z.B. bei Dateien, Auslesen des Speichers
- Operationszentriert: Operationen z.B. Dateien löschen, Software installieren, Prozesse anhalten

10.1.3 Meta-Berechtigungen

Pauschale Veränderung von Berechtigungen

10.2 Rechteverwaltung

10.2.1 Akteure einer Rechteverwaltung

- **Subjekt**: Wer?
- **Objekt**: Was kann bearbeitet werden?
- **Operation**: Was wird getan?
- Rechte pro **Subjekt**: *Capability*: Gutschein, der bestimme Operationen nur auf bestimmte Objekte erlaubt
- Rechte pro **Objekt**: *Access Control List* (ACL) Liste der Subjekte und deren ausführbaren Operationen
- Rechte pro **Operation**: Welche Subjekte dürfen Operation an welchen Objekten durchführen? →selten implementiert

10.2.2 Positive Rechte

Subjekt darf ohne Berechtigung nichts: Rechte werden mit Capabilities (für Prozesse) oder ACLs (für Dateien bzw. Verzeichnisse) hinzugefügt

10.2.3 Negative Rechte

Subjekt darf alles, solange nicht durch Capabilities oder ACLs eingeschränkt

10.3 Benutzerverwaltung

10.3.1 Gruppen

Problem: Rechtevergabe bei vielen Benutzern aufwendig →Einführung von Gruppen zur Steuerung von Rechten, erfordert Gruppenverwaltung

10.3.2 Anmeldung

Benutzer führt Anmeldung durch *Credentials* (Benutzername + Authentifizierung) durch, Abgleich durch Eintrag in sicherer Datenbank

10.4 Rechteverwaltung unter Linux / UNIX

10.4.1 Verwaltung von Subjekten / Objekten

- Inodes (5.3.1) als Objekt
- Benutzer als **Subjekt**, repräsentiert durch User-ID (UID) Nutzer 0 (root) hat vollständigen Zugriff
- Nutzergruppen als **Subjekt**: Gruppen werden eindeutig durch Group ID (GID) identifiziert

10.4.2 Berechtigungen von Inodes

- Eigentümer (UID)
- Gruppe (GID)
- Drei Berechtigungen: Eigentümer (**User**), Gruppe (**Group**), andere Nutzer (**Others**)

Rechte

- Datei: **r** (read), **w** (write), **x** (execute)
- Verzeichnis **r** (Auflisten des Verzeichnisinhalts), **w** (Löschen bzw. Anlegen von Verweisen), **x** (Auflösung von Pfadnamen)

Ausführung (execute)

- **User S-Bit**: Ausführung findet unter Berechtigung des Dateieigentümers statt, nicht unter Berechtigug des Nutzers
- **Group S-Bit**:
  - Dateien: Ausführung unter Gruppenberechtigung statt unter Gruppe des Nutzers
  - Verzeichnisse: neue Einträge bekommen automatisch selbe Gruppenzuordnung wie Verzeichnis
- **Sticky-Bit (T)**:
  - Dateien: Speicherseiten werden nicht ausgelagert →veraltet
  - Verzeichnisse: trotz Schreibrecht kann Nutzer nur eigene Einträge löschen z.B. für temporäre Dateien

10.4.3 Speicherung von Berechtigungen

Speicherung von Passwörtern

- /etc/passwd öffentlich lesbar, enthält Benutzername, IDs, Home-Verzeichnis, genutzte Shell
- /etc/shadow speichert Passwörter

Speicherung von Gruppen

- /etc/group öffentlich lesbar, enthält Gruppenname, ID, zugeordnete Nutzer
- /etc/gshadow enthälte Passwörter

10.4.4 Übetragung von Rechten

- Prozesstabelle enthält Berechtigungen zu Prozessen
- **Reale** UID, GID: Berechtigungen, unter denen der Prozess gestartet wurde
- **Effektive** UID, GID: Berechtigungen, die der Prozess momentan hat

Systemaufrufe zum Ändern der Berechtigungen: *setuid(), setgid()*

- Setzen reale & effektive ID eines Prozesses
- Nur root kann reale ID setzen

10.4.5 Erweiterte Berechtigungen

- Implementierung von POSIX-ACLs, sogenannte Extended Attributes
- Positive Rechte für Benutzer und Gruppen
- Eigentümer kann Rechte selbst setzen

Default-ACLs

- Beim Anlegen neuer Dateien werden Default-ACLs des Verzeichnisses übernommen
- Unterverzeichnisse übernehmen Default-ACLs

10.5 Rechteverwaltung unter Windows

10.5.1 Security Descriptors

- Dateien und Verzeichnisse sind Objekte
- Security Descriptors für jedes Objekt, ACLs sind an Security Descriptor gekoppelt

10.5.2 Rechtestufen

- **No access**: Kein Zugrff
- **List**: Anzeigen von Dateien in Verzeichnissen
- **Read**: Inhalt von Dateien lesen + List
- **Add**: Hinzufügen von Dateien zu einem Verzeichnis + List
- **Read & Add**: Read + Add
- **Change**: Ändern von Dateiinhalten, Löschen von Dateien & Read + Add
- **Full**: Ändern von Eigentümern und Zugriffsrechten & Change

10.5.3 Berechtigungen

- **SYSTEM** (Betriebssystem): Voller Zugriff
- **Administrator**: Kein vollständiger Zugriff
- Standardmäßige Vererbung der ACLs
- Negative Rechte überschreiben Positive Rechte